University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming deadlines:
  - Assignment 1 due on 3/14
  - Homework 7 due on 3/14
  - Lab 7 due on 3/18
  - Assignment 2 due on 3/28 (posted tonight)

Previous lecture …

- Single-pass fixed-codeword-size Compression
  - LZW compression and expansion

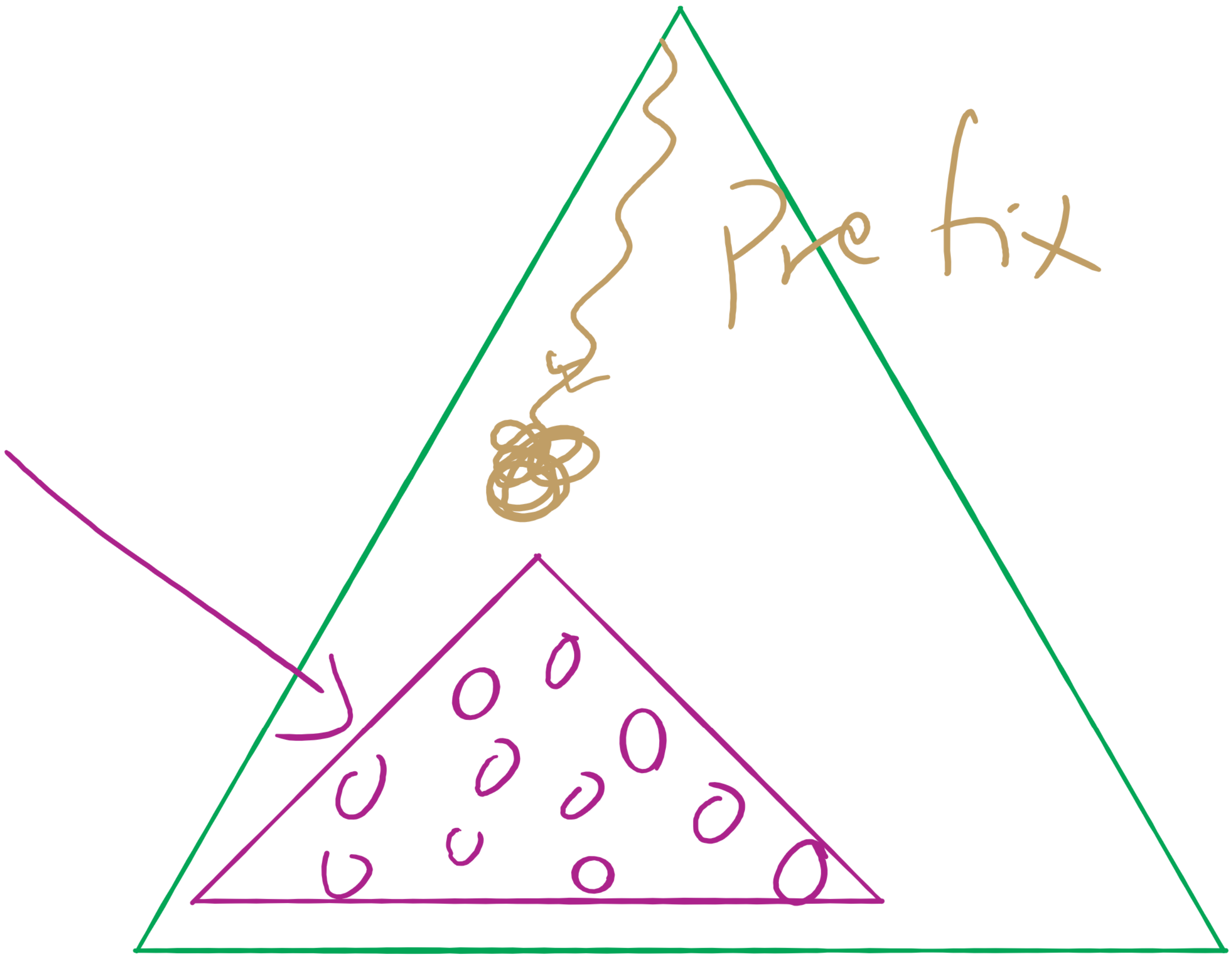
CourseMIRROR Reflections (Interesting)

- The variety of different compression algorithms and their advantages was most interesting in today's class.
- Going through examples of LZW compression
- Stepping through the LZW algorithm and thinking about how to store and extract as a data structure
- creating the code book during LWZ expansion
- How tries and sorted arrays are similar
- I found it interesting how LZW changes the bit size of the codewords
- The LZW examples were interesting to work through. (The quick reference to Weissman Score was funny, good to know.)
- I thought the test was pretty fair, not overly challenging
- The multiple questions for low points is nice to ensure a decent grade

CourseMIRROR Reflections (Confusing)

- The general approach to the LZW compression algorithm today was most confusing.
- the LWZ corner case
- Lzw and how a table is created
- i would like to review when to use LZW and when to use Huffman
- Im not sure I understand decoding using LZW for the case when the codeword is not in the codebook
- Please update the slide on GitHub before the lecture
- The test wasn’t awful, most difficult part being the LZW
- The true/false Java problem about the PHP Array (Lab6) runtime frequency analysis
- Some questions on the exam were confusing.

Assignment 1 runtime analysis



Problem of the Day

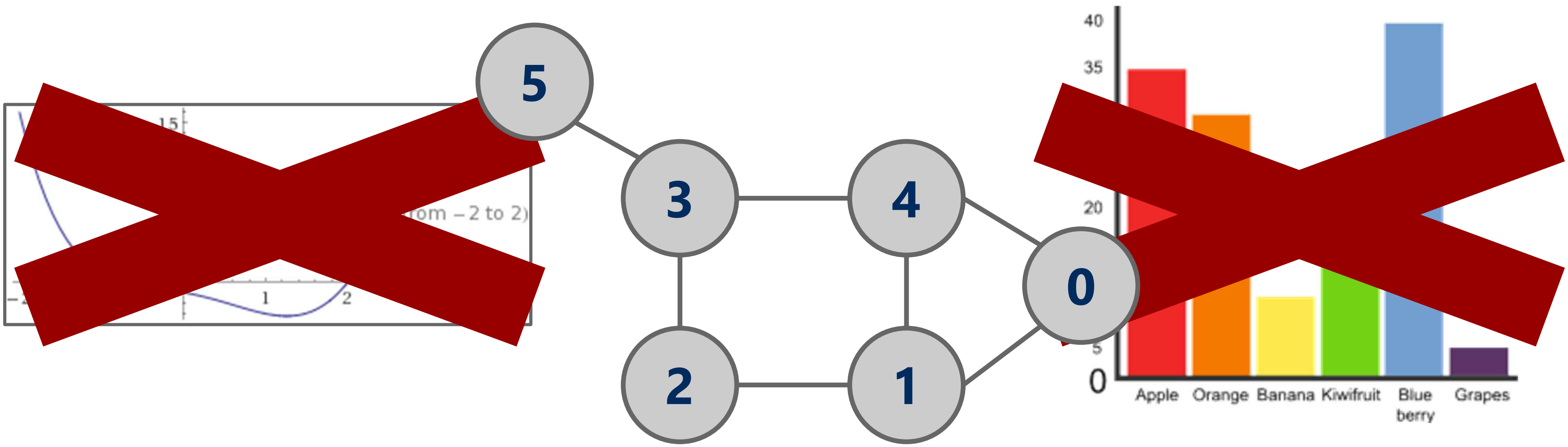
- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how "far" are they from each other?
    - E.g., 1^{st} connection, 2^{nd} connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

Which Data Structure to use?

Let's think first about how to structure the data that we have in memory.

- Account1: Connection1, Connection2, …
- Account2: Connection1, Connection2, …
- …

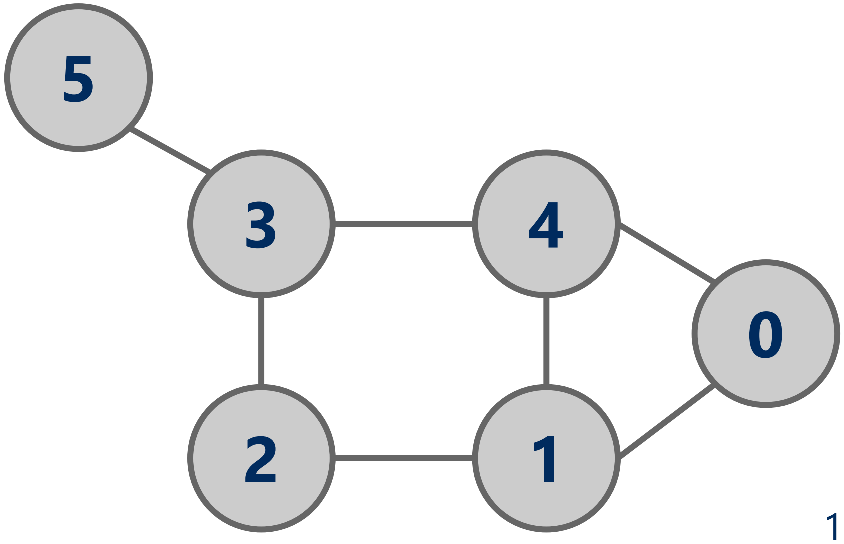
Graphs!



5
3
4
0
2
1
40
35
20
5
0
Apple
Orange
Banana
Kiwifruit
Blue berry
Grapes

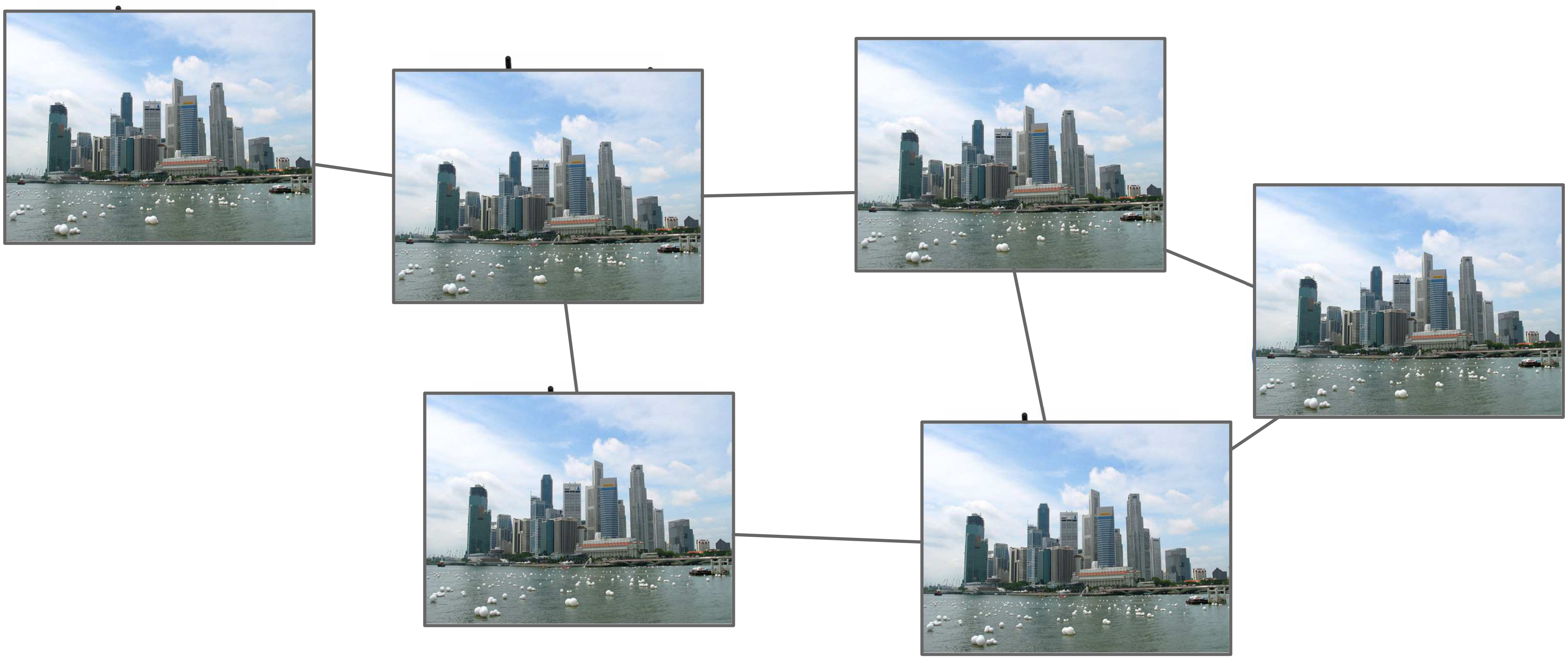
Graphs

- A graph G = (V, E)
  - Where V is a set of vertices
  - E is a set of edges connecting vertex pairs
- Example:
  - V = {0, 1, 2, 3, 4, 5}
  - E = {(0, 1), (0, 4), (1, 2), (1, 4), (2, 3), (3, 4), (3, 5)}



Why?

- Can be used to model many different scenarios

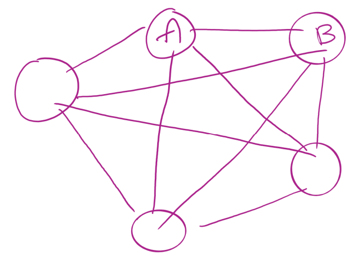


Some definitions

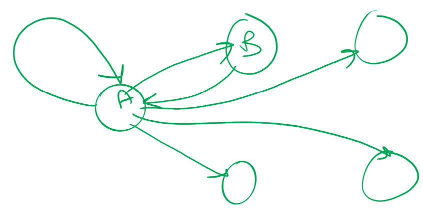
- Undirected graph
  - Edges are unordered pairs: (A, B) == (B, A)
- Directed graph
  - Edges are ordered pairs: (A, B) != (B, A)
- Adjacent vertices, or neighbors
  - Vertices connected by an edge

Graph sizes

- Let v = |V|, and e = |E|
- Given v, what are the minimum/maximum sizes of e?
  - Minimum value of e?
    - Definition doesn't necessitate that there are any edges…
    - So, 0
  - Maximum of e?
    - Depends…
      - Are self edges allowed?
      - Directed graph or undirected graph?
    - In this class, we'll assume directed graphs have self edges while undirected graphs do not



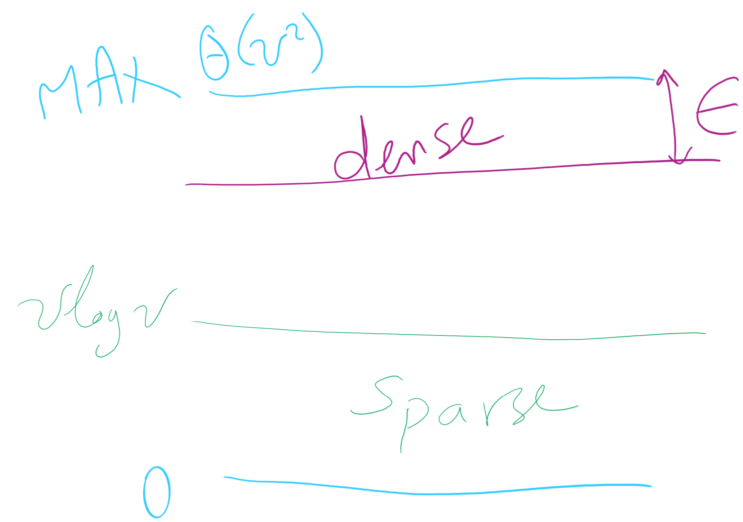
$$\frac{v(v-1)}{2}$$



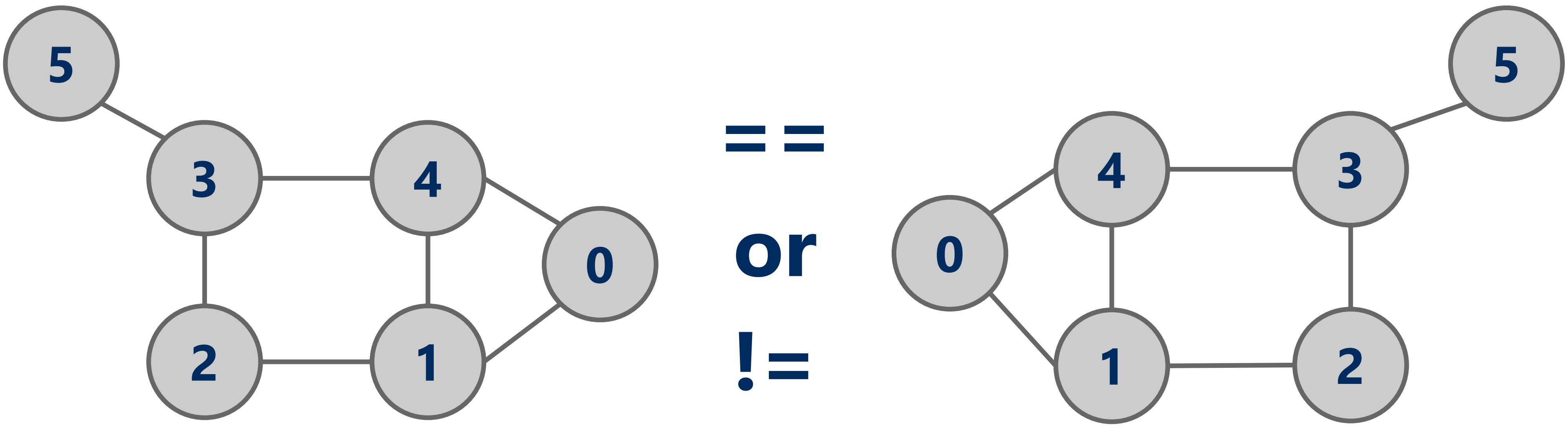
$$v * v = v^2$$

More definitions

- A graph is considered *sparse* if:
  - e <= v lg v
- A graph is considered *dense* as it approaches the maximum number of edges
  - I.e., e == MAX - ε
- A *complete* graph has the maximum number of edges



Question:



- ?

Representing graphs

- Trivially, graphs can be represented as:
  - List of vertices
  - List of edges
- Performance?
  - Assume we're going to be analyzing static graphs
    - I.e., no insert and remove
  - So what operations should we consider?

Using an adjacency matrix

- Rows/columns are vertex labels
  - M[i][j] = 1 if $(i, j) \in E$
  - M[i][j] = 0 if $(i, j) \notin E$

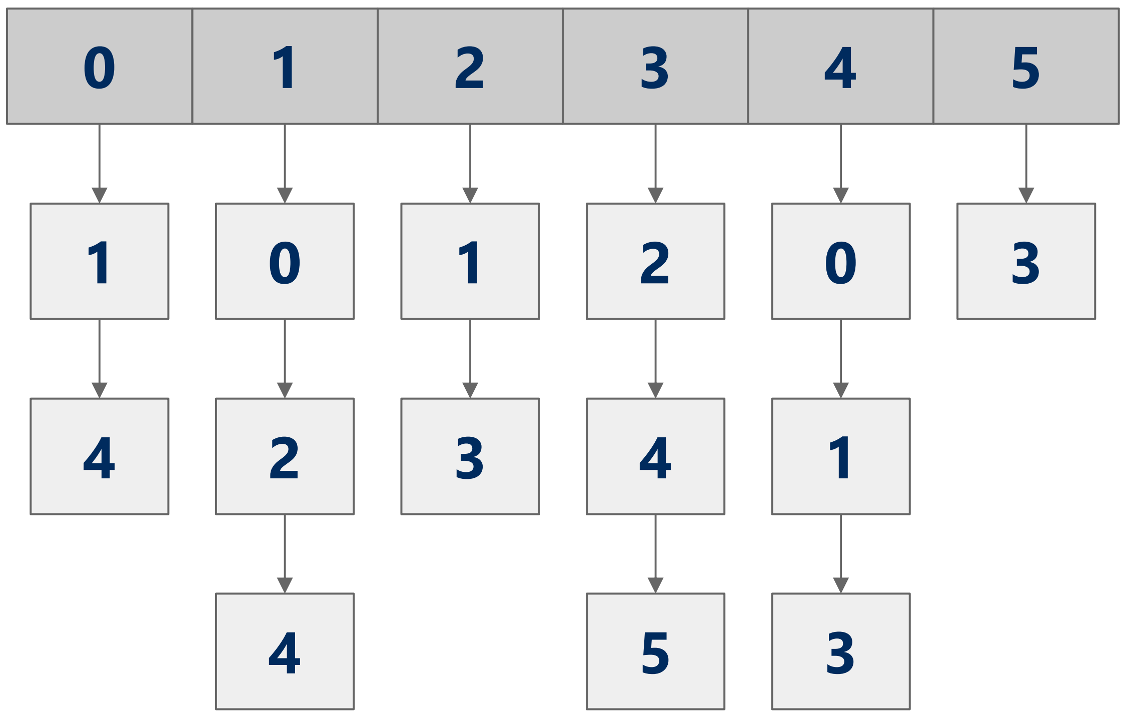
	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	0	1	0
2	0	1	0	1	0	0
3	0	0	1	0	1	1
4	1	1	0	1	0	0
5	0	0	0	1	0	0

Adjacency matrix analysis

- Runtime?
- Space?

Adjacency lists

- Array of neighbor lists
  - A[i] contains a list of the neighbors of vertex i



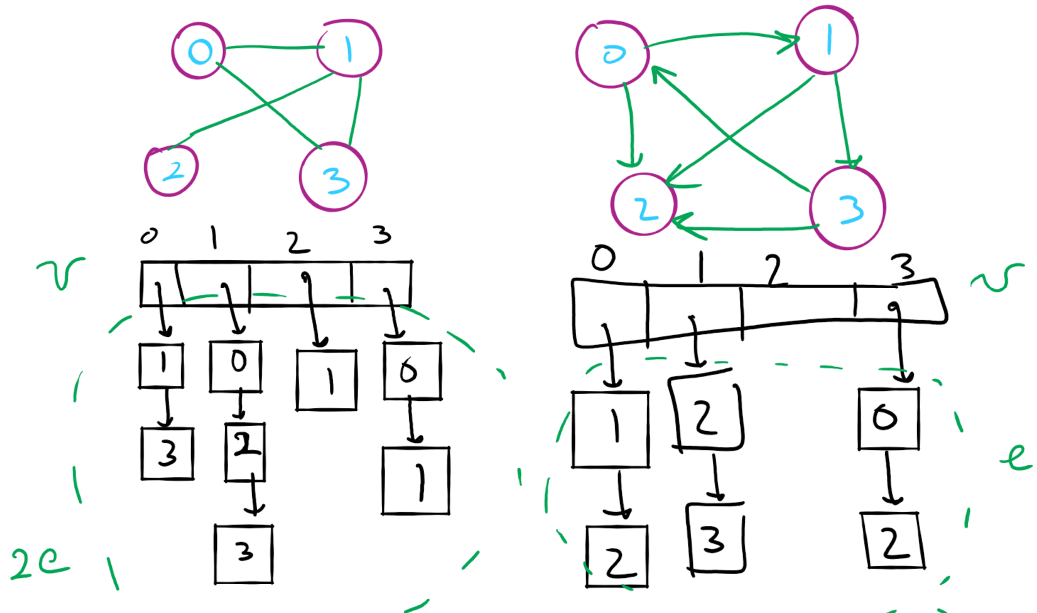
Adjacency list analysis

- Runtime?
- Space?

Comparison

- Where would we want to use adjacency lists vs adjacency matrices?
  - What about the list of vertices/list of edges approach?

Graph Representation Example 2



$\Theta(v+2e) = \Theta(v+e)$ $\Theta(v+e)$

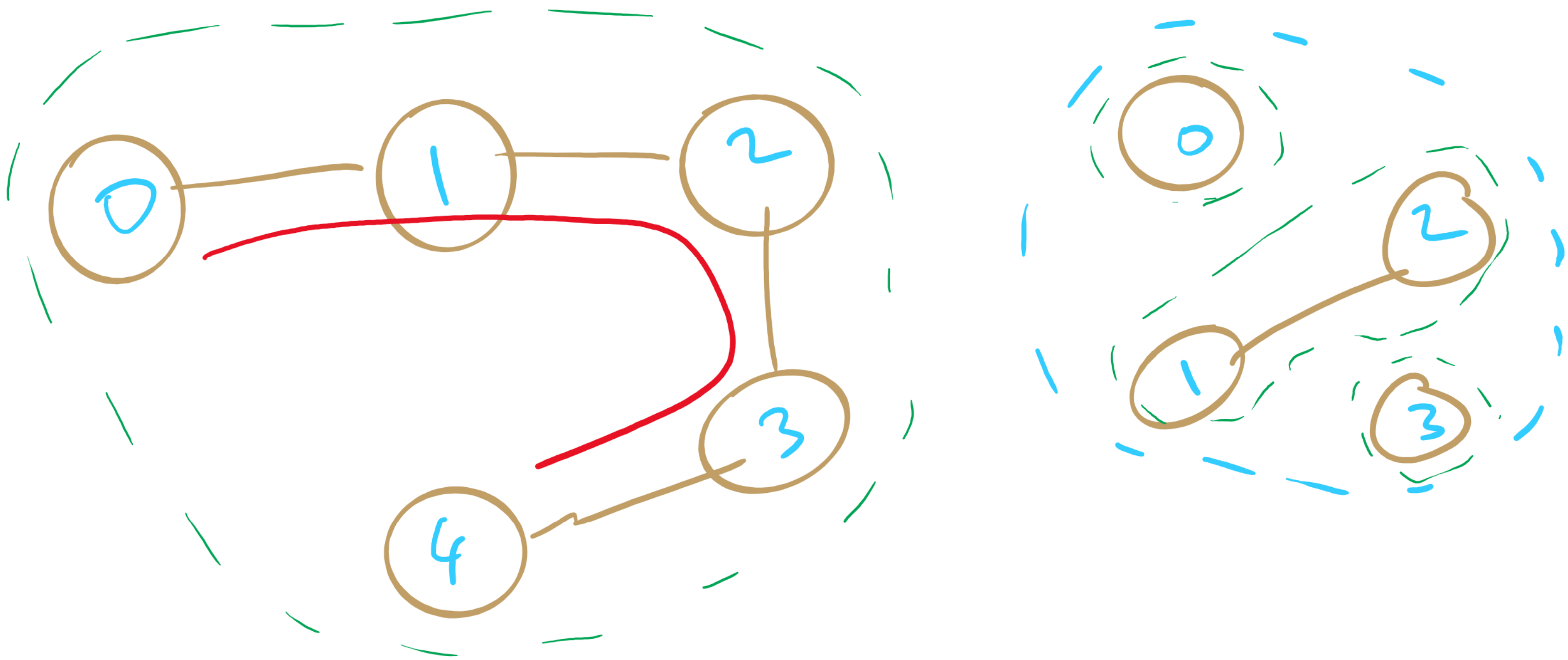
ArrayList<LinkedList<Node>> adjLists;

	0	1	2	3
0	0	1	0	1
1	1	0	1	1
2	0	1	0	0
3	1	1	0	0

$\Theta(v^2)$

	0	1	2	3
	0	1	1	0
	~~0~~	0	1	1
	0	0	0	0
	1	0	1	0

Sparse Graphs



Comparison

- Where would we want to use adjacency lists vs adjacency matrices?
  - What about the list of vertices/list of edges approach?

Adjacency Matrix vs. Adjacency Lists

	Checking if 2 vertices are neighbors	Retrieving list of neighbors	Size
Adj. Lists	$\Theta(\#\text{of neighbors})$	$\Theta(\#\text{of neighbors})$	$\Theta(v+e)$
Adj. Matrix	$\Theta(1)$	$\Theta(v)$	$\Theta(v^2)$

Even more definitions

- Path
  - A sequence of adjacent vertices
- Simple Path
  - A path in which no vertices are repeated
- Simple Cycle
  - A simple path with the same first and last vertex
- Connected Graph
  - A graph in which a path exists between all vertex pairs
- Connected Component
  - Connected subgraph of a graph
- Acyclic Graph
  - A graph with no cycles
- Tree
  - ?
  - A connected, acyclic graph
    - Has exactly v-1 edges

Graph traversal

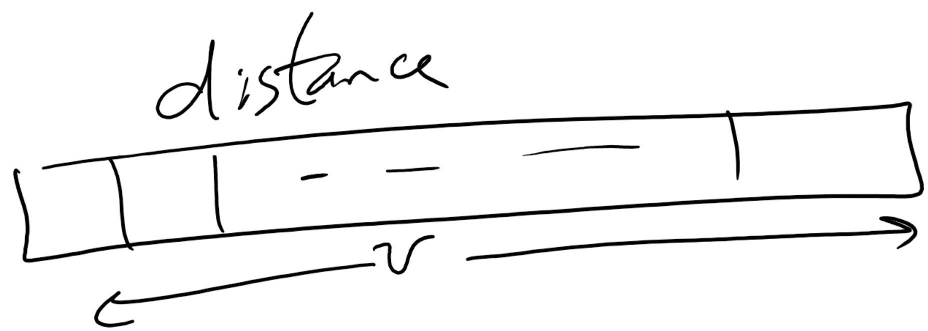
- What is the best order to traverse a graph?
- Two primary approaches:
  - Breadth-first search (BFS)
    - Search all directions evenly
      - I.e., from i, visit all of i's neighbors, then all of their neighbors, etc.
    - Would help us compute the distance between two vertices
      - Remember our problem of the day?
  - Depth-first search (DFS)
    - "Dive" as deep as possible into the graph first
    - Branch when necessary
    - Would help us find articulation points
      - Remember our problem of the day?

BFS

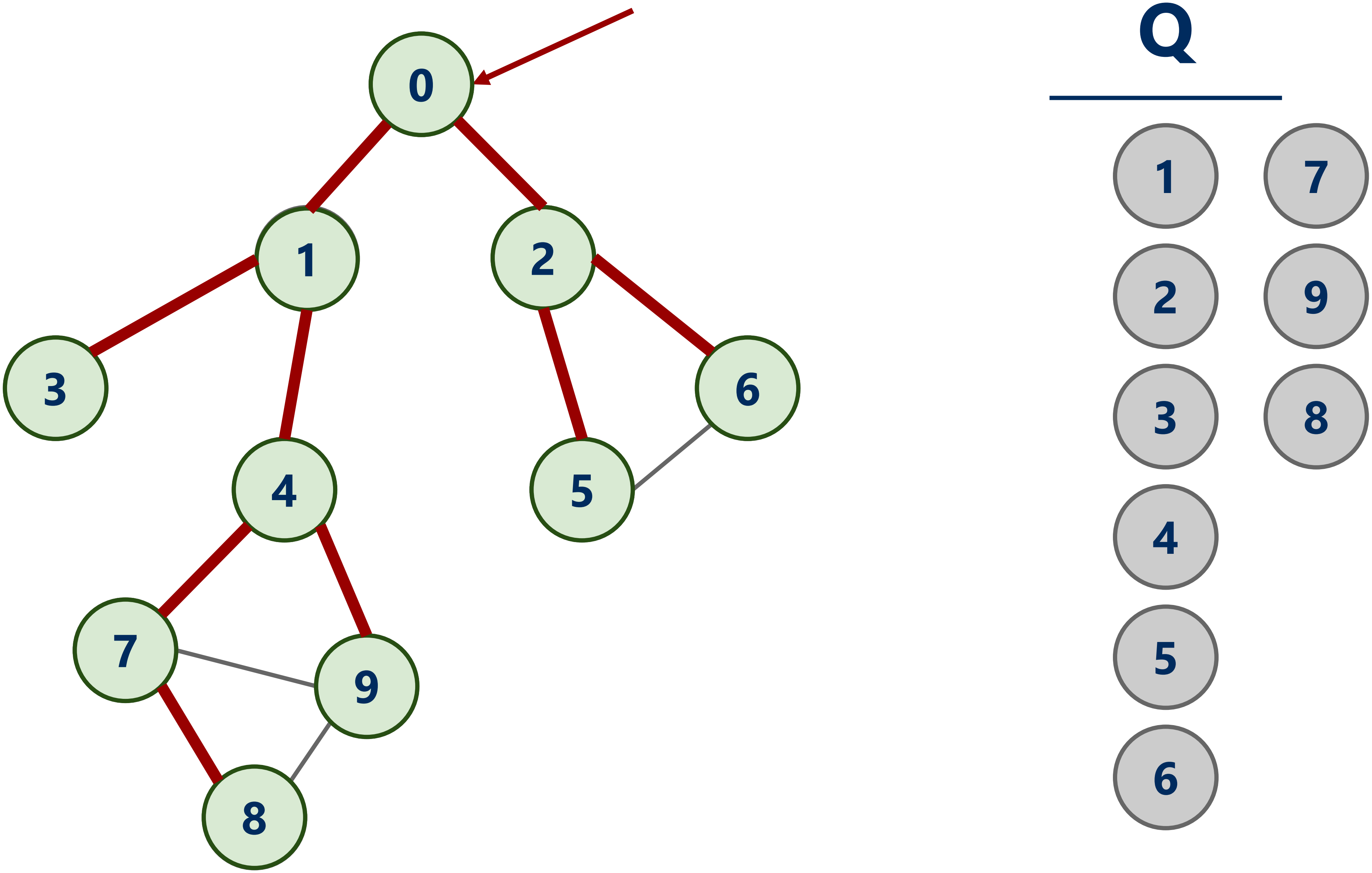
- Can be easily implemented using a queue
  - For each vertex visited, add all of its neighbors to the Q (if not previously added)
    - Vertices that have been seen (i.e., added to the Q) but not yet visited are said to be the *fringe*
  - Pop head of the queue to be the next visited vertex
- See example

BFS Pseudo-code

```
BFS ( vertex v ) {
    insert v into Q
    while ( Q is not empty ) {
        w = Pop head of Q
        visit w
        for each unseen neighbor x of w
            parent[x] = w; distance[x] = distance[w]+1
            add x to Q
    }
}
```



BFS example



Shortest paths

- BFS traversals can further be used to determine the *shortest path* between two vertices

Problem of the Day

- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how “far” are they from each other?
    - E.g., 1st connection, 2nd connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

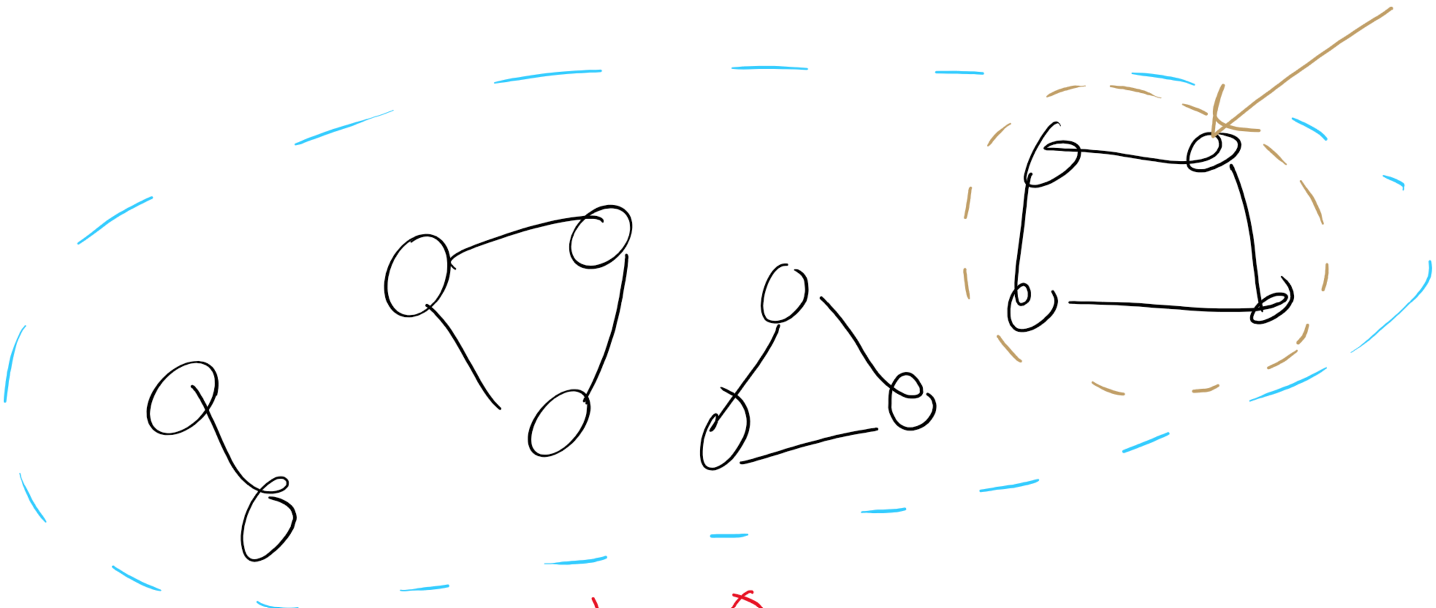
Problem of the Day

- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how “far” are they from each other?
    - E.g., 1st connection, 2nd connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

BFS would be called from a wrapper function

- If the graph is connected:
  - bfs() is called only once and returns a *spanning tree*
- Else:
  - A loop in the wrapper function will have to continually call bfs() while **there are still unseen vertices**
  - Each call will yield a spanning tree for a connected component of the graph

Wrapper function and connected components



int component = 0

for each vertex v

if v is unseen

DFS(v) / BFS(v)

component++

Wrapper function for BFS

int component = 0
for each vertex v in G
    if v not visited
        component++
        BFS/DFS(v)

component[v] = component

Problem of the Day

- **Input**: A file containing LinkedIn Connection information formatted like the following:
  - Account1: Connection1, Connection2, …
  - Account2: Connection1, Connection2, …
  - …
- **Output**: Answer the following questions:
  - Given two LI accounts, how "far" are they from each other?
    - E.g., 1st connection, 2nd connection, etc.
  - Are the accounts in the file all ***connected***?
    - If not, how many ***connected components*** are there?
  - Are there certain accounts that if removed, the remaining accounts become ***partitioned***?
    - These account are called ***articulation points***

DFS

- Already seen and used this throughout the term
  - For tries…
  - For Huffman encoding…
- Can be easily implemented recursively
  - For each vertex, visit first (in some arbitrary order) unseen neighbor
  - Backtrack at deadends (i.e., vertices with no unseen neighbors)
    - Try next unseen neighbor after backtracking

DFS Pseudo-code

DFS(Vertex v) {

// visit v

Mark v as seen

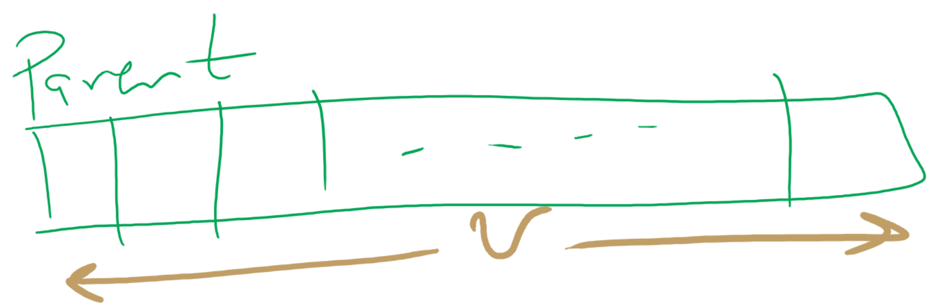
for each unseen neighbor w

Parent[w] = v

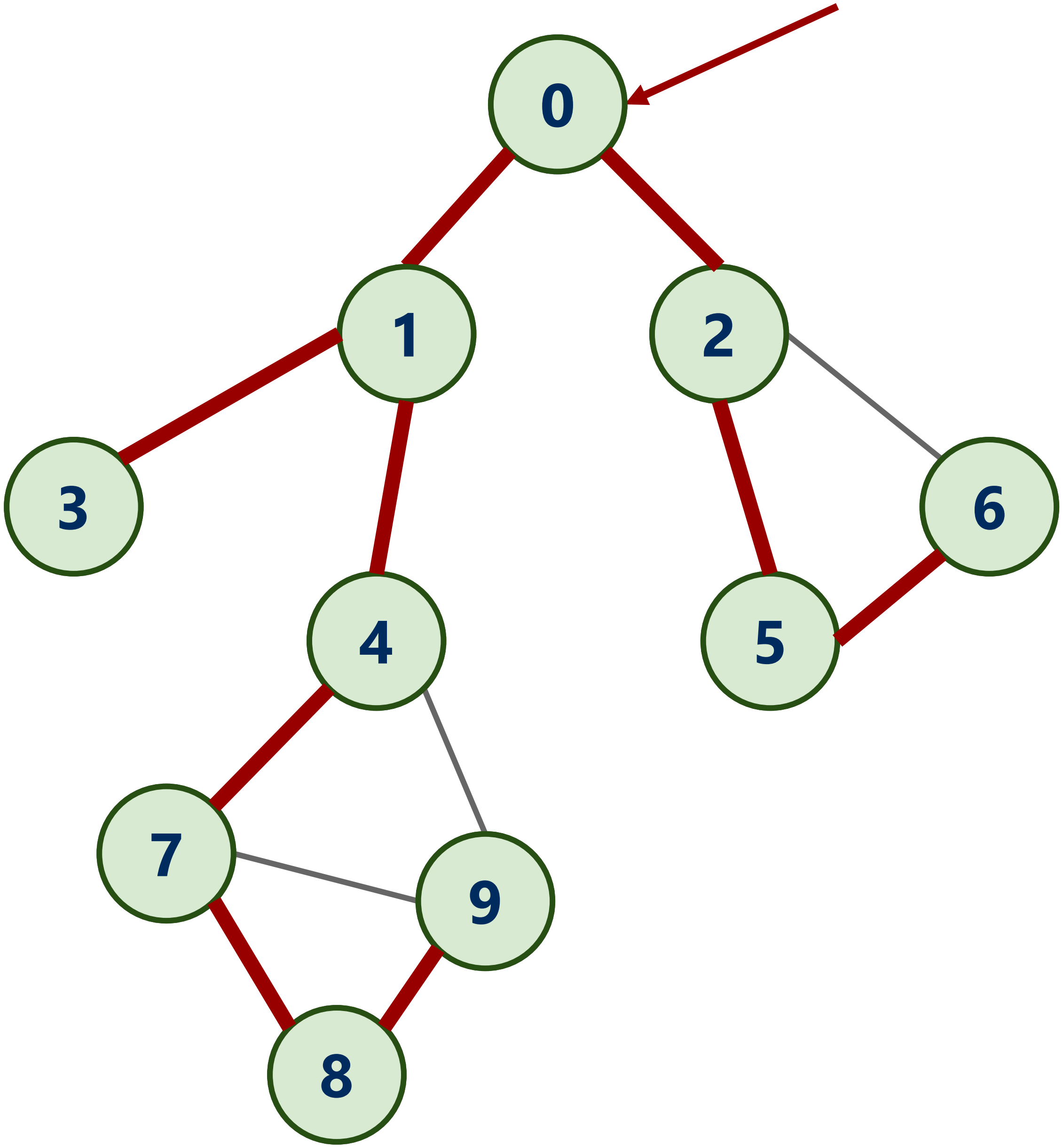
DFS(w)

// visit v

}



DFS example 2



Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com

8/29/2022